produced for it. I am afraid the document is somewhat long, 25 pages, so it will come out as a separate hand out.

3. Checker and Symbols Table

The checker takes the root of the syntax tree as a parameter and makes one or more passes over the tree. As the tree is traversed, the symbols table is built. This means that variables must be declared before they can be used and cannot be declared twice in the same block. In addition, as the symbols table is built, variables are given a unique rename value. Since there is only one data type, integer, we do not have to check that a floating point value is being assigned to a Boolean variable, for example. However, we do need to check that a void function is not being used in an expression, that a void function does not have a return statement within it, and that a non-void function contains a return statement. It is not necessary to check if the return statement in a non-void function is reachable, nor is it necessary to check that formal and actual parameters match in type and quantity. However, when an array value is used and if the array is defined in the same block, the value is checked to see if it is in bounds.

Since the symbols table grows and shrinks as blocks are opened and closed, for this project, create a second table. When an item is added to the symbols table, add it to this duplicate table as well; however, this duplicate table does not shrink when the symbols table does. At the end of this phase, print out this table. In this way, all items that were added to the symbols table can be checked.

What items should be in the symbols table? The following is my definition.

```
public String ID;                 //The lexeme
public int entryType;             //variable, array, etc.
public int dataType;              //INT or VOID
public int blockLevel;            //The nesting level: 0 is lowest level
public TreeNode parameterList;    //Just copied from the syntax tree
public int returnType;            //For functions: INT or VOID
public int arrayMax               //The size of an array
public String rename;             //Each variable is given a unique name
```

4. Code Generator

The target language for this compiler is VM4 assembly language. In this manner, the VM4 assembler can translate the object file into a VM4 executable file that can be executed by the VM4 simulator. However, the code generator can be the most difficult phase of a compiler to write; so, I will give you a class file called `CodeGen.class`. The constructor for this class takes two objects, the first is the root of the syntax tree (a `TreeNode` object) and the second is a `File` object into which the output will go. The `File` object should be opened before being passed into the constructor and it should refer to a file with an `.asm` extension. The method to generate the code is `public` and is called `genCode()`. For this phase of the project, you just